Scheduling Working Group
Scheduling Request for Comments:8.5
Category: Informational

J.M. Schopf
Northwestern University

July 2001

Ten Actions When SuperScheduling

Status of this Draft

This draft provides information for the grid scheduling community. Distribution of this memo is unlimited.

Abstract

This document has three purposes. The first is to outline how a user currently approaches the problem of scheduling across multiple administrative domains. Please note that while these are grouped and numbered they do not in any way refer to an ordering. The second purpose of this document is to show where a small amount of effort could make a large amount of change and make life easier for the application scientist. The third is to annotate where work related to this is being performed, and what state it is currently in.

1. Introduction

SuperScheduling has been defined as the process of scheduling resources where that decision involves using multiple administrative domains. Several different scenarios fall under this domain: searching multiple administrative domains to use a single machine; speculatively submitting a job to single machines at multiple sites where all but one of the submissions is canceled at a later time; scheduling a single job to use multiple resources at a single or multiple site.

We leave the definitions of “job” and “resource” purposely vague at this time to avoid religious arguments. A job can be anything that needs a resource – from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep). A resource is anything that can be scheduled, a machine, disk space, some QoS network, a person, etc.

We use the word “step” and a numbering system for easy reference. This does not imply that these actions are actually performed in this order, or that the all MUST occur in every system that is modeled after this approach. In general, don’t pay too much attention to the numbering, this is just one possible ordering. Some of the steps may be iteractive, recursive, repeated, or just plain ignored.

One of the primary differences between a superscheduler and other common schedulers is that the superscheduler does not own the resources and therefore does not have total control over them. Furthermore, the superscheduler does not have control over the entire set of jobs on the system, or even necessarily know about them, so decisions about an entire set of jobs to a resource cannot be made. This lack of ownership and control are the sources of many of the problems to be solved in this area.

The most common current superscheduler is the user. There are however several efforts underway to change this [NAB99, Silver, PBS, Loadleveler, LSF]. This document walks through the steps that a user currently (June 2000-July 2001) goes through when scheduling a job over resources on multiple administrative domains. In Section 3 examples of some of these steps are given with respect to specific system implementations.

2. Stages of SuperScheduling

A user goes through three stages to schedule a job when it involves multiple sites. Phase one is resource discovery, in which the user makes a list of potential resources to use. Phase two involves gathering information about those resources and choosing a best set to use. In phase three the user then runs the job.

Examples of current approaches to aid users in these steps are given in Section 3.

Phase 1: Resource Discovery

Resource discovery involves the user selecting a set of resources to investigate in more detail in phase two, information gathering. At the beginning of this phase, the potential set of resources is the empty set, and at the end of this phase, the potential set of resources is some set that has passed a minimal feasibility requirement. Most users do this in three steps: authorization filtering, job requirement knowledge, and filtering to meet the minimal job requirements.

Step 1 Authorization Filtering

It is generally assumed that a user will know which resources he or she has access to in terms of basic services.

At the end of this step the user will have a list of machines or resources to which he or she has access.

Step 2 Application requirement definition

In order to proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources (see Step 3).

The set of possible job requirements can be very broad, and vary significantly between jobs. It may include static details, such as the operating system or hardware for which a binary of the code is available, or that the code is best suited to a specific architecture. Dynamic details are also possible, for example a minimum RAM requirement, connectivity needed, /tmp space needed, etc. This may include any information about the job that should be specified to make sure that the job can be matched to a set of resources.

Step 3 Minimal requirement filtering

Given a set of resources to which a user has access and the minimal set of requirements the job has, the third step in the resource discovery phase is to filter out the resources that do not meet the minimal job requirements. The user generally does this step by going through the list of resources and eliminating the ones that do not meet the job requirements as much as they’re known. It could also be combined with the gathering of more detailed information about each resource (step 4) (and in fact this is how most proposed systems go about the process). However, when being done by hand, if a user can eliminate an inappropriate resource it is done at this stage to simplify the information gathering in the next phase.

Phase 2 System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or

single resource set) must be selected on which to schedule the job. This is generally done in two steps: gathering information and making a decision.

Step 4 Gathering information (query)

In order to make the best possible job/resource match, a user needs to gather dynamic information about the resources in question. Depending on the application and resource in question, different information may be needed.

Take for instance the simple case of finding the best single resource for a job to run on. A user might want to know the load on the various machines, or the queue lengths if the machine has queues. In addition, physical characteristics and software requirements play a role – is the compiler you need on the machine, is the disk big enough for the data, etc. Then there are location/connectivity issues – is the machine close enough to the data store?

All of these issues are multiplied in the case of multiple resources.

Making an advance reservation (see Step 6) may or may not be a part of this step.

Step 5 Select the system(s) to run on

Given the information gathered by the previous step, a decision of which resource (or set of resources) should the user submit a job is made in the step. This can and will be done in a a variety of ways.

Note that this does not address the situation of speculative execution – when a job is submitted to multiple resources, and when one begins to run the other submissions are cancelled. This is only the selection of a resource (or set of resources).

Phase 3 Run job

The third phase of superscheduling is running a job. This involves a number of steps, few of which have been defined in a uniform way between resources. They include:

Step 6 (optional) Make an advance reservation

It may be the case that to make the best use of a given system, part or all of the resources will have to be reserved in advance. Depending on the resource, this can be easy or hard to do, may be done with mechanical means as opposed to human means, and the reservations may or may not expire with or without cost.

Step 7 Submit job to resources

Once resources are chosen the application must be submitted to the resources. This may be as easy as running a single command or as complicated as running a series of scripts, and may or may not include setup or staging (see Step 8).

Step 8 Preparation Tasks

The Preparation stage may involve setup, staging, claiming a resveration, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at NASA was considered unsuccessful because it did not address the need to stage files automatically, for example.

Step 9 Monitor progress (maybe go back to 4)

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing.

Step 10 Find out J is done

When the job is finished, the user needs to be notified.

Step 11 Completion tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, break down the environment, remove temporary settings, etc.

3. Examples

Several parts of the process described in Section 3 have begun to be addressed by current systems. In this section we address this work.

Phase 1: Resource Discovery

Step 1 Authorization Filtering

The current common solution to knowing where one is for the user to have a list of user names, machine names, and passwords. While this is a security risk if someone else finds the list, that access can be controlled by the user in most cases (by, for example, locking it in the top drawer of a desk). While the information is generally available when needed, this method is known for problems with fault tolerance.

There has been some discussion on how this could be done automatically. One way to accomplish this is with a PKI infrastructure with single-sign on to simplify the pre-requisite authentication. Note that this is not neceesarily true for the authorization. The authorization issue by itself is quite important, and is not addressed here.

Alternatively, this information could be kept as part of a central information service with other system information [FFK+97]. This approach is currently being used by the KB scheduler from Poznan [Nab99], but has scalability problems.

Another solution is the use of “Smart Cards”, which are credit card-like devices that hold account information for a user on them. However, this assumes that a central agency has agreed on how to issue smart cards between administrative domains, and the smart card readers have been installed wherever access is needed.

Step 2 Application requirement definition

Currently, while the need for this information is recognized, for example with AppLeS [Apples, BW96, BWF+96], the Network Weather Service [Nws,WSH99], Condor ClassAds [RLS00, RLS98] etc., it is generally assumed in most system work that the information is simply available. Very little work has been done to automatically gather this data, or to store it for future use. This is in part because the information may be hard to discover.

Attempts to have users supply this information on the fly has generally resulted in data that has dubious accuracy - for example, notice how almost every parallel scheduler requires an expected execution time, but almost every system administration working with these schedulers compensates for the error in the data, by as much as 50% in some [ASW99]. Attempts to gather the information, for example by using forms, has been found to be extremely time consuming as well [GFUWG99].

In the future, one can envision compilers that could aid in supplying basic requirements for applications, or monitoring.

Step 3 Minimal requirement filtering

A resource description language (Condor's ClassAds[Condor, RLS00, RLS98], Globus' RSL**[CFK+98, FFK+97]**, etc.) in combination with some Grid Information Service (GIS) could be used to perform this filtering, but has not yet to our knowledge.

Phase 2 System Selection

Step 4 Gathering information (query)

Ideally, a user (or scheduling service) would ask "If I give this job to a resource (or set of resources), how long will it take to run?" Howeverm this is an unsolved question. Instead, one area that is low-hanging fruit would be to have an interface for each scheduler/resource management system to query "If I give you a job that looks like XX, when will it start?" [SC00].

This doesn't answer the real question a user would like to ask, namely, when will the job END, but it approximates it. There is a large body of predictive work, but most of it requires additional information not available on current systems.

There are various interfaces to data sources to help aid users collect some of this information by hand. Most recent in these is the Globus/NCSA Grid Searcher Project [Gridsearcher] which gives a user-friendly front-end to the Globus MDS information service. Similarly, the Alliance User Portal [AUP] and the Npaci Hotpage [hotpage] gather together some of the needed information for the user.

Step 5 Select the system(s) to run on

One system that could be extended to address some of this is the Condor Matchmaker/classad system [Condor, RLS00, RLS98], which matches resources to jobs based on user defined ranking equations

Phase 3 Run job

Step 6 (optional) Make an advance reservation

A current example of making an advance reservation involves a user calling a system admin to reserve time on a large machine for demo purposes.

Several projects have concentrated on advance reservations [Legion, Silver, PBS]. Additional work has been done by Roy [GARA, FRS00] in reserving resources other than machines or networks.

Step 7 Submit job to resources

A typical example of a user doing this is running the qsub. For example:

```
qsub -l ncpus=4,walltime=2:00:00 MyJob
```

Step 8 Preparation tasks

Currently, a user will run scp or ftp to assure that the data files needed are in place.

Several groups have begun to address this problem, one being Globus with their High Throughput Broker (htb) [CFK+98]. HTB had scripts set up to submit a set of directories to each needed machine and do the pre-staging for the user automatically. Currently in Globus, GASS [GASS] provides staging.

Condor [Condor] also supports a variation of this where it actually runs an extra process before and after the application is run to do staging and clean-up.

Unicore [Unicore] can also support automated pre-staging.

Step 9 Monitor progress (maybe go back to 4)

Today, this is typically done by repetitively querying the resource for status information

Step 10 Find out J is done

Often, submission scripts for parallel machines will include an email notification parameter.

Step 11 Completion tasks

Currently, this is done most often by hand by the user using scp or ftp. Many of the current systems that do staging (Step 8) als o handle cleanup.

4. Conclusion

This document defines the steps a user follows to make a scheduling decision across multiple administrative domains at the present time.
We also review current approaches in these areas.

5. References

[Apples] Application Level Scheduling (AppLeS), http://apples.ucsd.edu/

[ASW99] Alliance Scheduling Workshop, UIUC, February, 1999.

[AUP] Aliance User Portal (AUP), http://aup.ncsa.uiuc.edu/

[Condor] Condor, http://www.cs.wisc.edu/condor/publications.html

[BW96] Fran Berman and Rich Wolski, “Scheduling from the Perspective of the Application”, Proceedings of the Symposium on High Performance Distributed Computing, 1996 (also available at http://apples.ucsd.edu/pubs/hpdc96.ps).

[BWF+96] Fran Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao, “Application-Level Scheduling on Distributed Heterogeneous Networks”, Proceedings of Supercomputing 1996 (also available at http://apples.ucsd.edu/pubs/sup96.ps and UCSD CS Tech Report #CS96-482) .

[CFK+98] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, “Resource Management Architecture for Metacomputing Systems”, Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998 (also available at ftp://ftp.globus.org/pub/globus/papers/gram97.ps)

[FFK+97] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, “A Directory Service for Configuring High-Performance Distributed Computations” Proc. 6th IEEE Symp. on High-Performance Distributed Computing, pg. 365-375, 1997 (also available at

http://ftp.globus.org/pub/globus/papers/hpdc-97-mds.pdf).

[FRS00] I. Foster, A. Roy, V. Sander, “A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, (8th International Workshop on Quality of Service, 2000) (also available at http://www.globus.org/documentation/incoming/iwqos_adapt1.pdf)

[GARA] Globus Advance Reservation Architecture, http://www-fp.mcs.anl.gov/qos/papers/gara_admin_guide.pdf

[GASS] Global Access to Secondary Storage (GASS), http://www.globus.org/gass/

[GFUWG99] Grid Forum Users Working Group, www.gridforum.org, 2000.

[Gridsearcher] Sean Melody and Jennifer Schopf, http://anchor.cs.nwu.edu

[Hotpage] Hotpage, https://hotpage.npaci.edu/

[Nab99] Jarek Nabrzyski, Knowledge-based Scheduling Method for Globus, Globus Retreat, Redondo Beach, 1999, http://www.man.poznan.pl/metacomputing/ai-meta/globusnew/index.html

[Legion] Legion, http://legion.virginia.edu/

[LSF] LSF, http://www.platform.com/products/LSF/

[Loadleveler] Loadleveler, http://www-1.ibm.com/servers/eserver/pseries/software/sp/loadleveler.html

[NWS] The Network Weather Serrvice, http://nws.npaci.edu/NWS/

[PBS] Portal Batch System (PBS), http://pbspro.com

[RLS00] Rajesh Raman, Miron Livny, and Marvin Solomon, "Resource Management through Multilateral Matchmaking", Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania, August 2000, pp 290-291. (also available at http://www.cs.wisc.edu/condor/doc/gangmatching.ps)

[RLS98] Rajesh Raman, Miron Livny, and Marvin Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing", Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL(also available at http://www.cs.wisc.edu/condor/doc/hpdc8.ps)

[SC00] Snell and Clement, "Metascheduling Query and Reservation Interface", Global Grid Forum Schedule working Group WD 2.2, April 2000, http://www.cs.nwu.edu/~jms/sched-wg/WD/schedwd2.2.pdf

[Silver] Silver, http://www.supercluster.org/projects/silver/

[Unicore] Unicore, http://www.unicore.de/

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes, “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”, Journal of Future Generation Computing Systems, 1999, (also available at http://www.cs.ucsd.du/users/rich/papers/nws-arch.ps.gz or as UCSD Technical Report Number TR-CS98-599, September, 1998)

6. Author’s Addresses

Jennifer M. Schopf

Computer Science Department
Northwestern University
1890 Maple Ave.
Evanston, IL 60201
Tel: (847)491-7320
Fax: (847)491-5258
jms@cs.nwu.edu